Next Generation Parallel Machines

Speaker: Bill Broadley, Geology, UC Davis
bill@cse.ucdavis.edu

Topics:

- Computing Trends
- Parallel Technologies
- Cluster Design

Time: 3:30-4:00

Why Parallel?

The Good:

- Faster

The Bad:

- Often requires rewriting code
- Parallel code is much more complex, algorithms harder to improve, and functionality harder to extend.
- Parallel code is extremely hard to debug, 64 node cluster runs 1 trillion instructions a second, 1 quadrillion in 16 min.
- Reliability drops (checkpoints help)
- Scaling is tough (see Amdahl's law)
- Interconnects are expensive

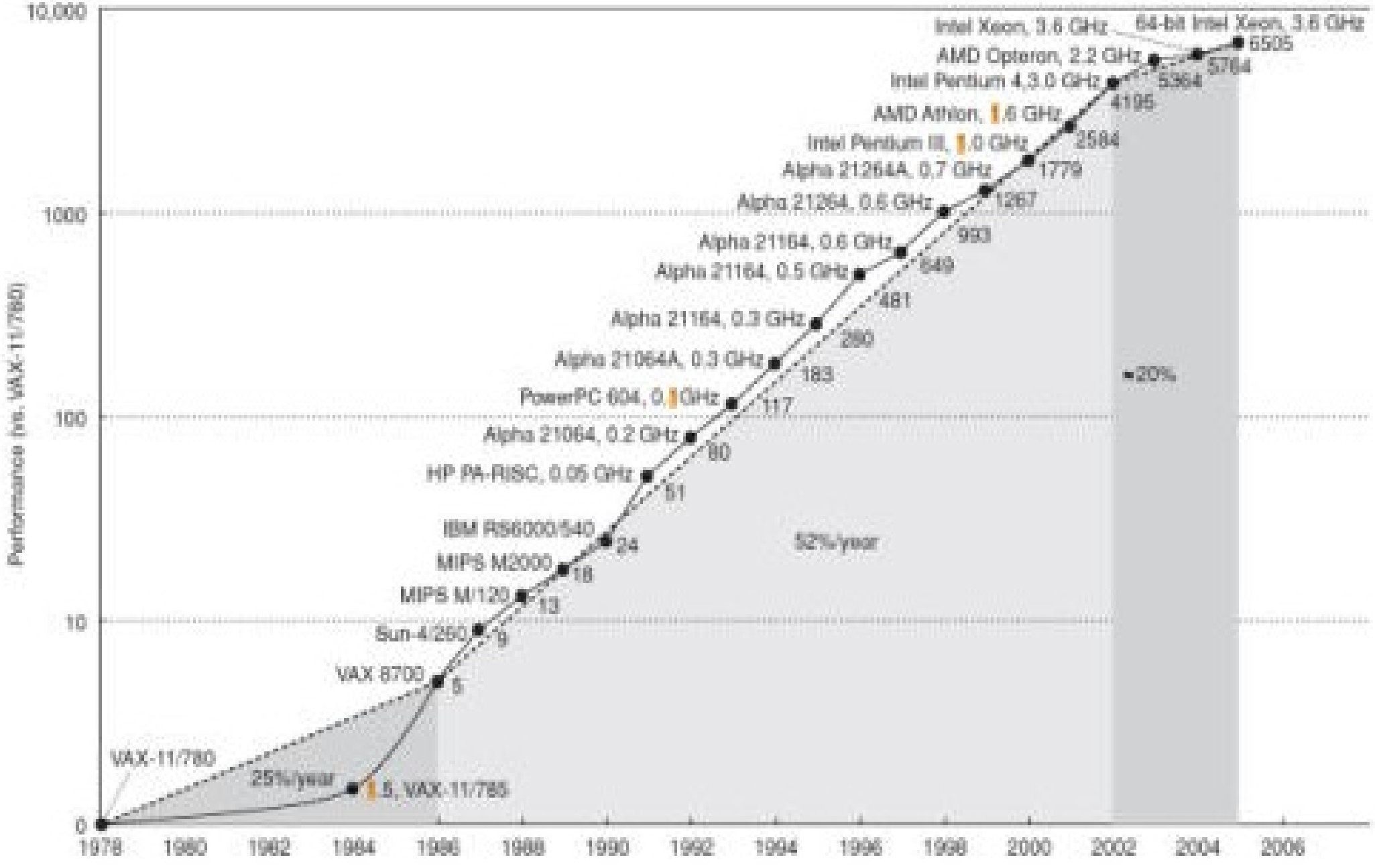
Why Parallel?

Can't we wait? Aren't computers getting twice as fast every 2 years?

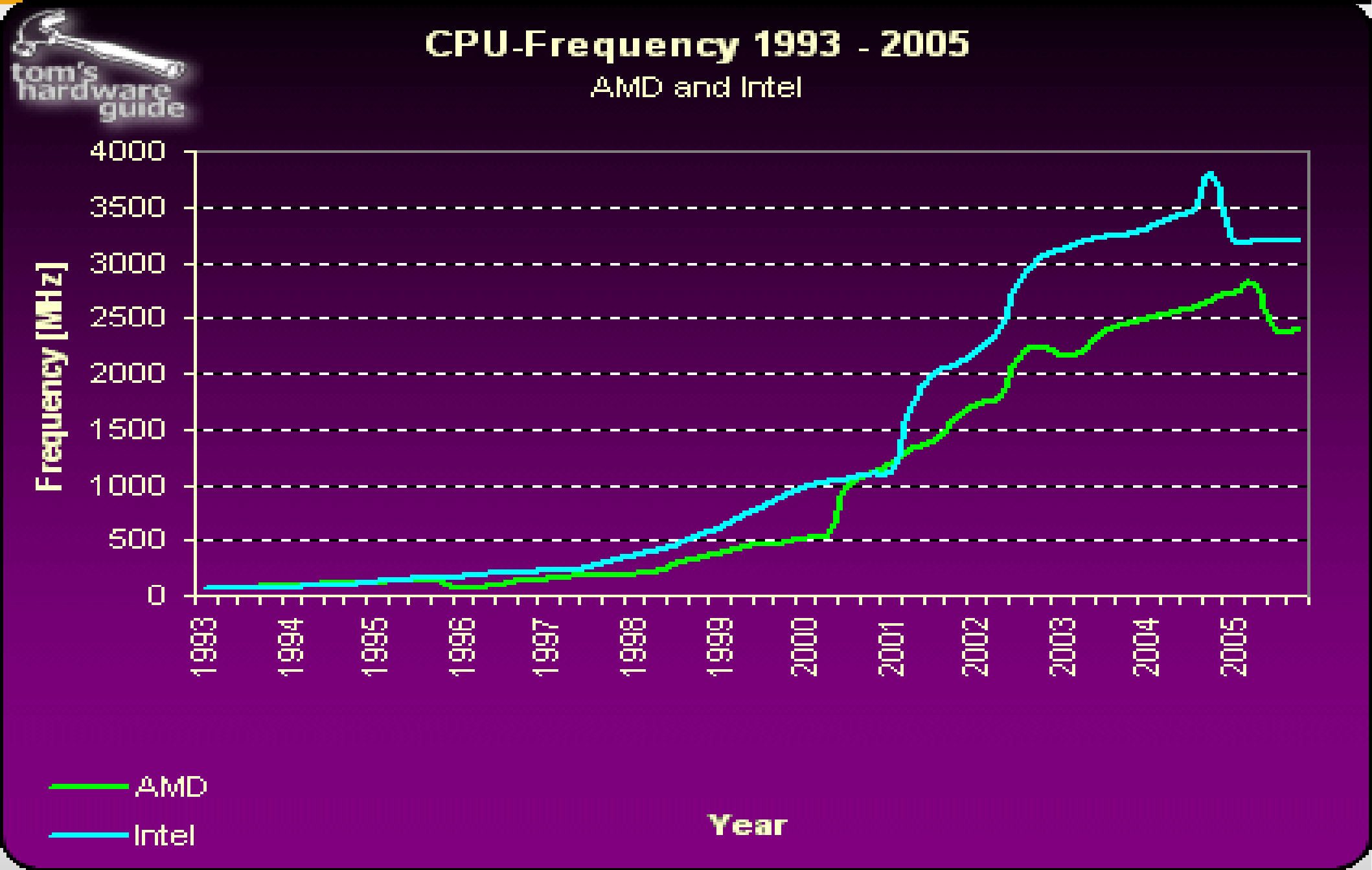
... after 16 years of doubling performance every 18 months, single-processor performance improvement has dropped to modest annual improvements. This fork in computer architecture means that for the first time in history, no one is building a much faster sequential processor. If you want your program to run significantly faster ... you're going to have to parallelize your program.

[Computer Architecture, Fourth Edition: A Quantitative Approach]

Why Parallel?



Clock speed


CPU-Frequency 1993 - 2005
AMD and Intel
tom's hardware guide
Frequency [MHz]
4000
3500
3000
2500
2000
1500
1000
500
0
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
Year
AMD
Intel

Traditional parallelism within a node

SMP - Hardware

- Quad core desktops are under $1,000
- Dual socket quad core servers start around $2k
- Quad socket quad core servers start around $7k

Software

- Shared memory – explicity sharing memory between processes.
- Pthreads – new processes that share all memory
- MPI – sending messages between processes.
- OpenMP – C and Fortran support for taking loops that are loop independent to use multiple cores on a shared memory machine

Current/Near future parallelism within a node

Software

- MPI – Message Passing Interface, the default for clusters and HPC
- CUDA – Nvidia GPU language
- Intel Threaded Building blocks – C++ libraries for threading
- Google's Map/Reduce and Yahoo's Hadoop.

Hardware

- Nvidia GTX 280 – 140GB/sec, 1GB, ~1TF, and 240 cores
- Intel Larabee – 32 cores, 512 bit (16 floats or 8 doubles), ~2TF
- Intel Nehalem – 8-12 cores, ~3GHz, 25-30GB/sec
- Future is clearly > 1000s of cores, and slower single threads

CUDA

- Single and (very recently) Double precision support.
- 128 core model with 512MB and 60GB/sec start at $160
- Memory is limited to 512MB (cheap) to 1GB (expensive)
- arXiv.org lists 3 papers, all mention impressive speedups:
  - 80x, 40x, and 120x on MD simulations
  - 100x on an N-body simulation
  - 120x on Fast K nearest Neighbor searchs
- This was BEFORE the latest gen GPUs with double precision.

How hard is this node parallel programming?

Lets look at a[i]=b[i]+scalar*c[i]

OpenMP:

```
#pragma omp parallel for
    for (j=0; j<N; j++)
        c[j] = a[j]+scalar*b[j];
#endif
```

Cuda:

```
STREAM_Triad<<<dimGrid,dimBlock>>>(d_b, d_c, d_a, scalar,  N);

__global__ void STREAM_Triad( float *a, *b,*c, float scalar, int len)
{
  int idx = threadIdx.x + blockIdx.x * blockDim.x;
  if (idx < len) c[idx] = a[idx]+scalar*b[idx];
}
```

Thread Building Blocks Parallel_For

```
task_scheduler_init init(nthread);
parallelOperation(a, b, c, nValues);

void parallelOperation( float* a, float* b, float* c, size_t n){
    Parallel par;
    par.a = a; par.b = b; par.c = c;
    parallel_for( blocked_range<int>( 0, n), par,
auto_partitioner());
}

class Parallel {
public:
    float* a;    float* b;    float* c;
    void operator( )( const blocked_range<int>& range ) const{
        for ( int i=range.begin(); i!=range.end( ) ; ++i )
            a[i]=b[i]+c[i]*1.5;
    }
};
```

How scalable is a node's memory system?

Sometimes Great (AMD):
1 thread(s), a random cacheline per 99.54 ns, 99.54 ns per thread.
2 thread(s), a random cacheline per 50.03 ns, 100.05 ns per thread.
4 thread(s), a random cacheline per 25.44 ns, 101.74 ns per thread.
8 thread(s), a random cacheline per 12.21 ns, 97.70 ns per thread.

Sometimes less so (Intel):
1 thread(s), a random cacheline per 102.65 ns, 102.65 ns per thread.
2 thread(s), a random cacheline per 57.41 ns, 114.82 ns per thread.
4 thread(s), a random cacheline per 31.46 ns, 125.84 ns per thread.
8 thread(s), a random cacheline per 27.65 ns, 221.19 ns per thread.

Bandwidth, AMD (MB/sec):

Threads	Rate
1T Copy:	4,913
2T Copy:	9,515
4T Copy:	10,593
8T Copy:	14,160

Intel (MB/sec)

Threads	Rate
1T Copy:	4,077
2T Copy:	6,090
4T Copy:	6,244
8T Copy:	6,134

128 thread Nvidia run:

Function	Rate (MB/s)
Copy:	50,077
Scale:	50,637
Add:	51,090
Triad:	50,527

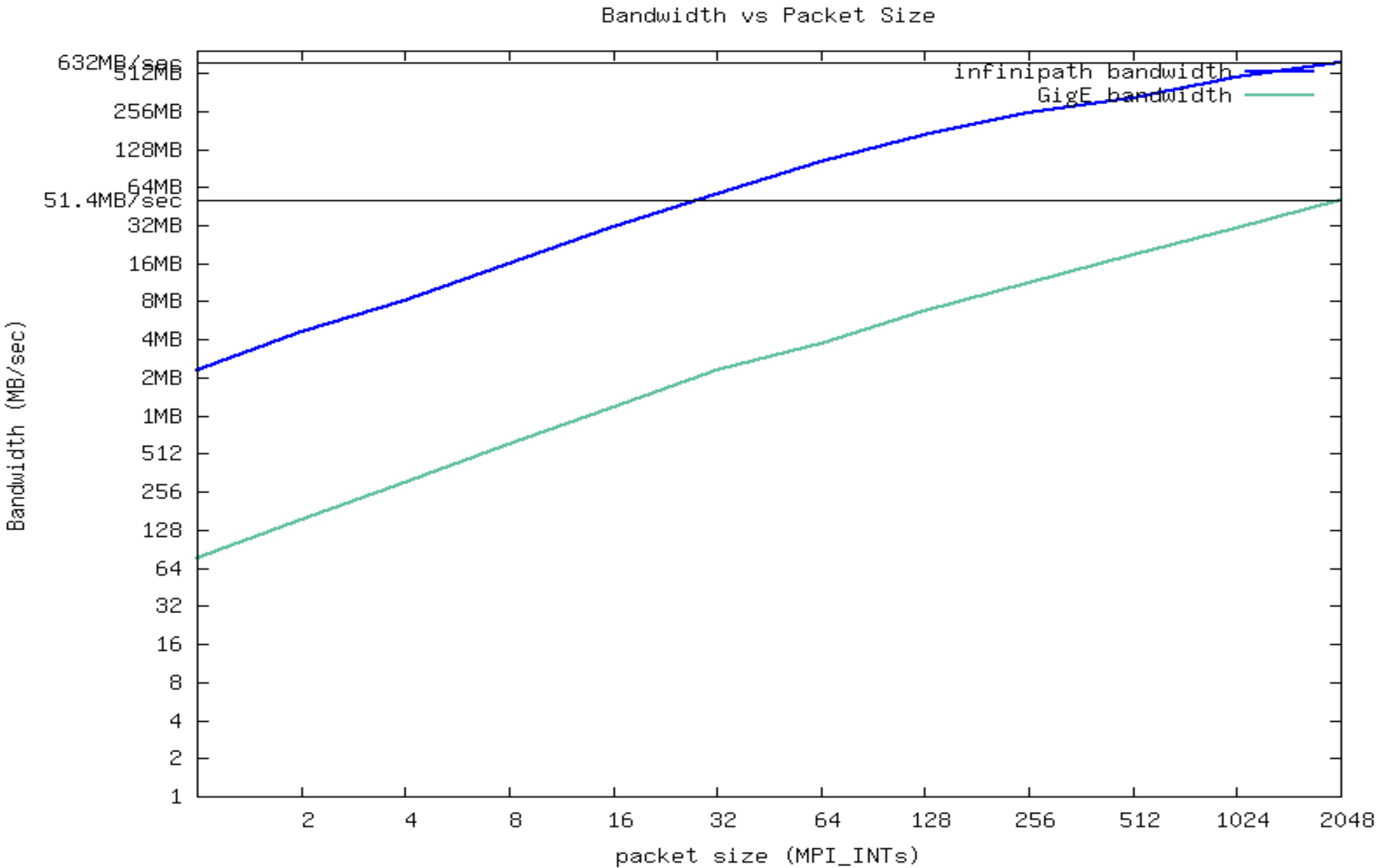
Cluster interconnect technologies

- GigE – Cheap, at least at low port counts, 1 Gbit, 25-50 us latency
- IB – Expensive, 8 Gbit, 2-4us latency
- Infinipath – Expensive, 8 Gbit, 1.5us latency
- Infinipath DDR/IB-DDR – Expensive, 10-12 Gbit, same latency.
- Myrinet 10G – Expensive, 10 Gbit, 2.5us.

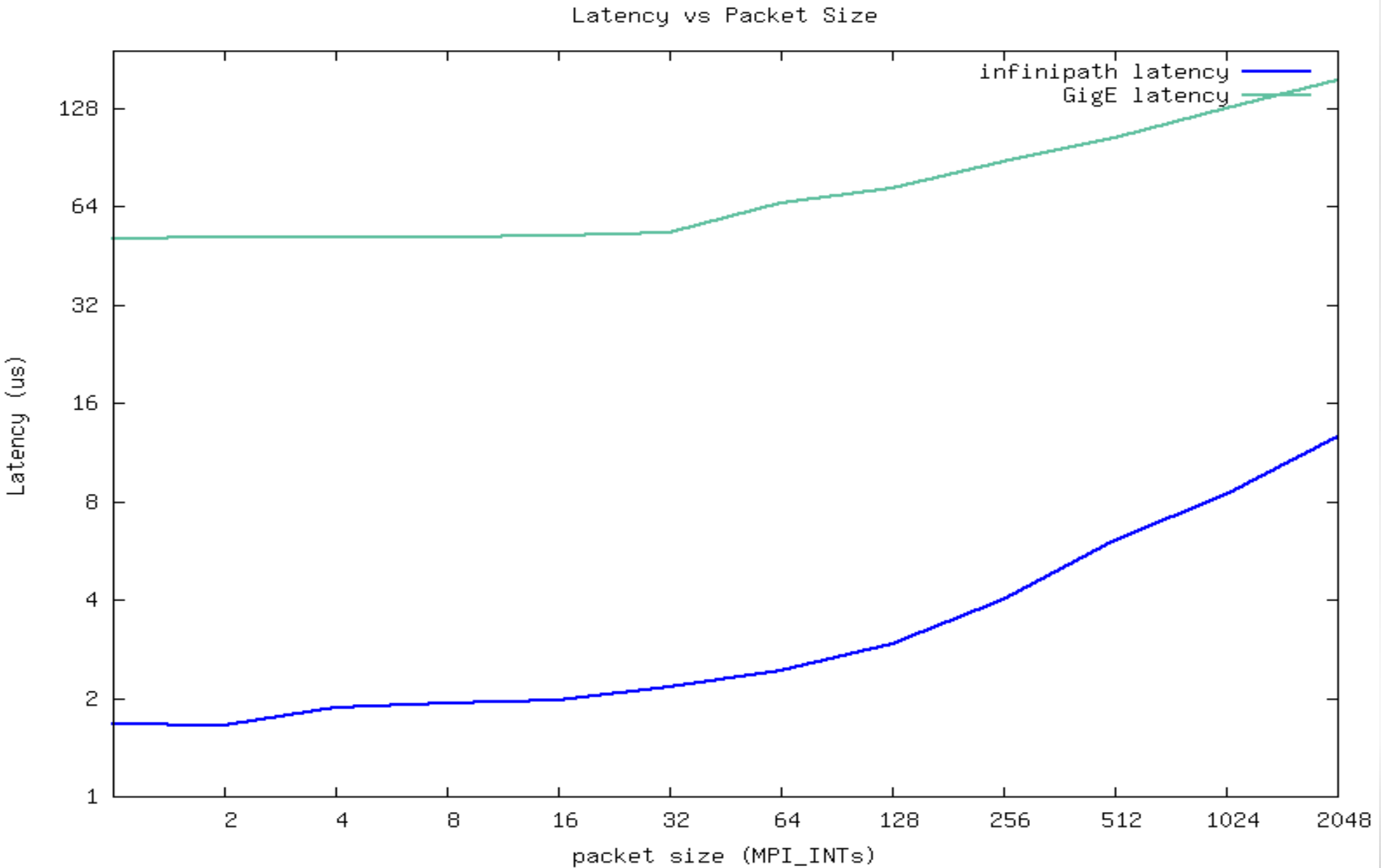
Why pay for an expensive interconnect?

- Ideally you don't. They are expensive, fragile, require specialized skills and increase administration costs. Often 1/3 to ¼ of the cluster budget.
- Latency improvements of 25x are not uncommon.
- Codes commonly scale to 2-4x the size of GigE clusters.
- If you can keep most communication within a node, GigE will likely work well.
- Negative scalling with GigE is common.
- There are natural cluster sizes for all interconnects, the step function in cost and complexity can be substantial. 25 ports of IB requires 5 24 port switches and 49 cables (or a 96/144 port switch, even a half full 96 port IB switch ~= $60k)

GigE vs Infinipath Bandwidth



GigE vs Infinipath Latency



Cluster Planning Considerations

- Actual delivered cooling needs to handle the building/room heat load and 100% of the electrical load (which will be turned into heat). A kill-a-watt is very handy for measuring actual power used.
- Unlike administrative computing, clusters often run processors at 100% and power circuits at 80%. Measure, report, and design accordingly.
- Weight - 48 1U machines in a rack can exceed weight/density building limits.
- Vibration and Noise – Can exceed occupational limits, even through walls.
- Power distribution – 208V 3phase via bus bars is recommended if available.
- Management – IPMI, switched PDUs, KVM over IP, and phased power on encourages higher availability.
- Fileserver design – SAN, iSCSI, direct attached, parallel (PVFS or Lustre). Currently 16 disk fileservers are working well for my clusters.
- UPS (usually for fileservers only), backups.
- Switches, expandability, cabling harness, console, compilers distribution, and warranty.

Benchmark Basics

- Do not look at clock speeds, look at application performance.
- Do not look at peak or theoretical memory bandwidth, look at relevant size and pattern, or at least McCalpin's stream. Avoid 2 DDR2*667 = 10GB/sec.
- Do not look at link speeds (GigE, 10G) look at measured bandwidth for a relevant packet size and communications pattern.
- Do not look at disk bus speeds, but performance for a relevant record size and access pattern, or at least the disk head rate.

Benchmarking

- Your application is best, ideally with a real dataset using the number of CPUs, nodes, and array sizes of a production run.
- Scaled down runs (array size, nodes, or CPUs) are often not representative.
- Microbenchmarks are useful, but easy to misinterpret. The interconnect with the lowest latency and highest bandwidth does not necessarily have the best application performance.
- McCalpin's stream benchmark is good for measuring node memory bandwidth and OpenMP scaling.
- SPECfp_rate2006 is good for measuring CPUscaling.
- Pstream is good for mapping out today's parallel memory hierarchies.
- Beware of shared caches and memory systems.

Examples

- Intel 3.33 Ghz SpecFP = 24.3 is faster than an Opteron 2.3 Ghz = 19.4 for single thread... Intel looks good.
- Opteron dual socket 2.5 Ghz = 90.1 is faster than an Intel 3.4 Ghz at 62.3... AMD looks good.
- Opteron quad socket 2.5 Ghz = 165 is faster than an Intel 2.93 Ghz at 119... AMD looks good.

Alas, unless you are buying hardware to run Spec instead of applications, making purchasing decisions based on Spec is dangerous. Even if your application performs similar tasks the performance correlation is likely to be poor.

SpecCPU FP 2006

- Contains 17 real scientific applications with input, output, verifyable answers, and hairy code paths. No tiny loops that don't touch memory (Mflop, Mips, Drystones, bogomips).
- Includes: Quantum Chemistry, several CFDs, Molecular Dynamics, Finite Element Analysis, Linear Programming, Ray-tracing, Electromagnetics, Quantum Chemistry, Weather, Speech recognition, and General Relativity codes.
- Compilers and flags make a big difference, useful for shopping for compilers and/or finding good flags to use.
- Runs with a single thread and parallel for single thread performance and throughput, the later is most cluster relevant.
- Numbers are available to allow calculations of the most interesting subset of applications

Dangerous Generalizations

- Intel quads have large caches and are faster at floating point for cache friendly codes.
- AMD has a better memory system, can retrieve a random cacheline every 11ns, has 17GB/sec bandwidth and is faster when running cache unfriendly codes.
- Intel quads worsen the already poor memory situation of the intel duals (no faster than the intel single socket).
- Codes will vary widely, top, ps, and memory used during a run has little to do with cache friendliness (or lack of).
- Intel has plans to fix their memory problem later this year with Nehalem.

Spec Results

SPEC® CFP2006 Result

Copyright ©2007 Standard Performance Evaluation Corporation

Dell Inc.

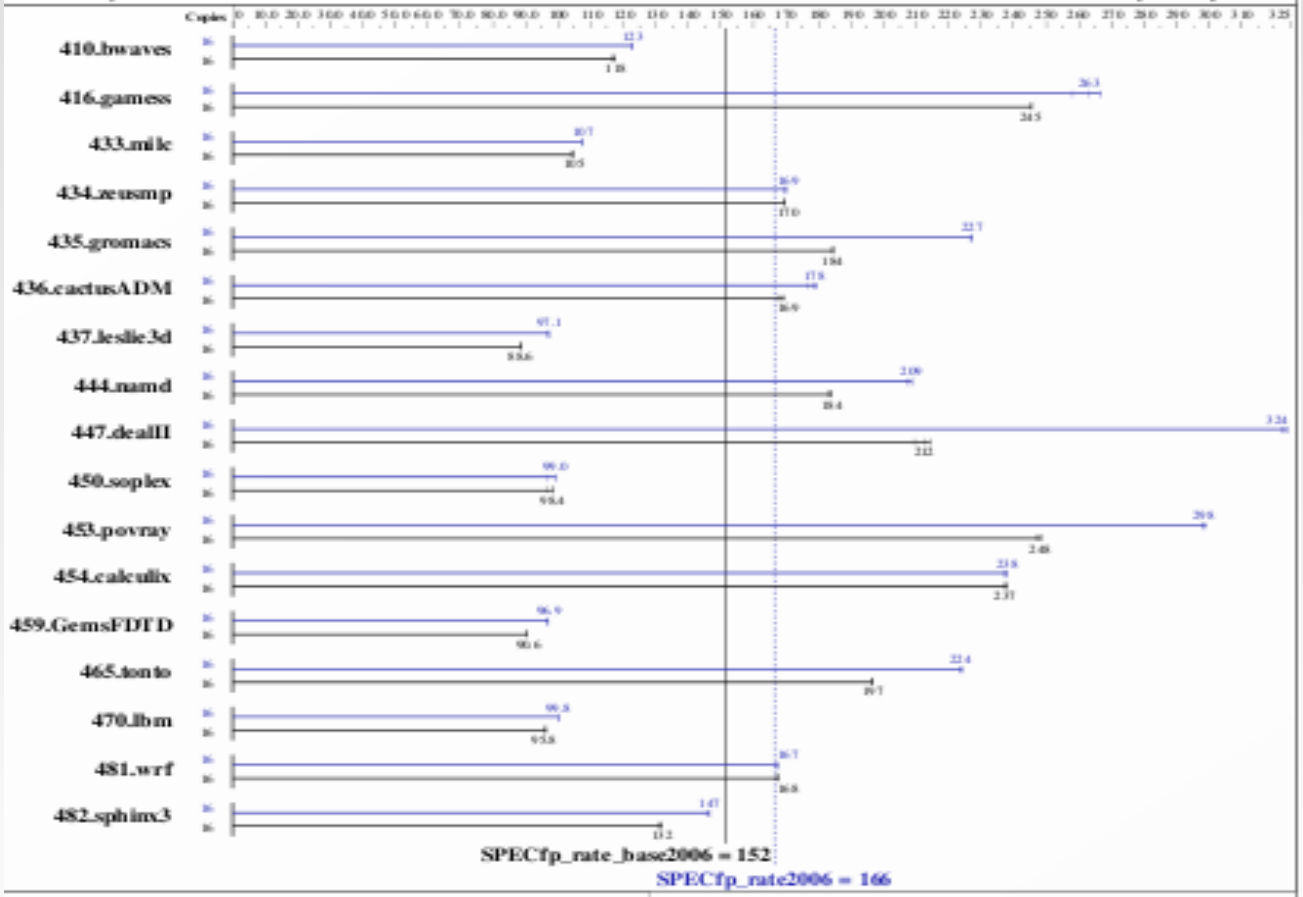
PowerEdge R905, AMD Opteron 8360 SE (2.50 GHz)

SPECfp®_rate2006 = 166

SPECfp_rate_base2006 = 152

CPU2006 license: 55
Test sponsor: Dell Inc.
Tested by: Dell Inc.

Test date: Apr-2008
Hardware Availability: Jun-2008
Software Availability: May-2008

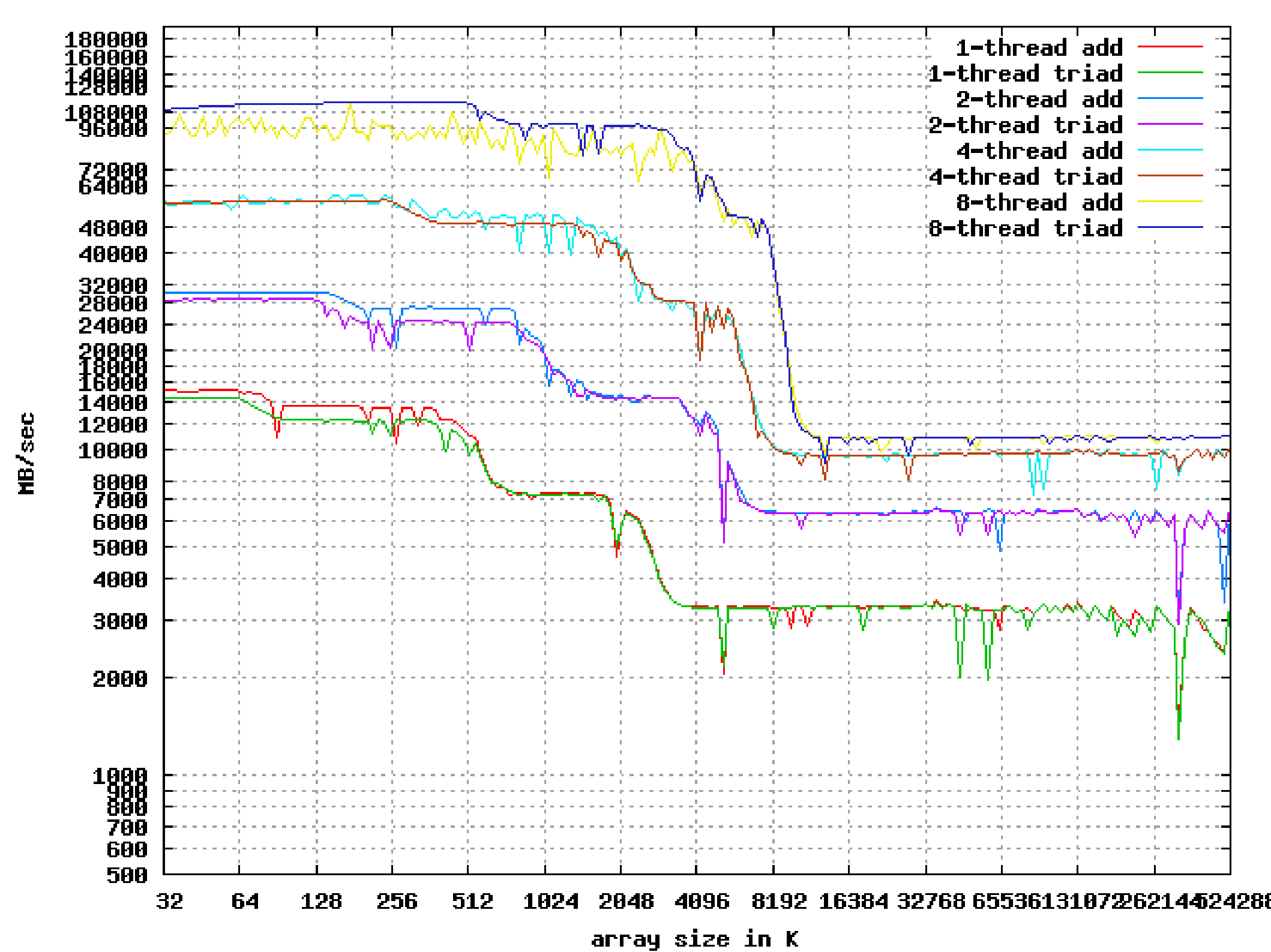


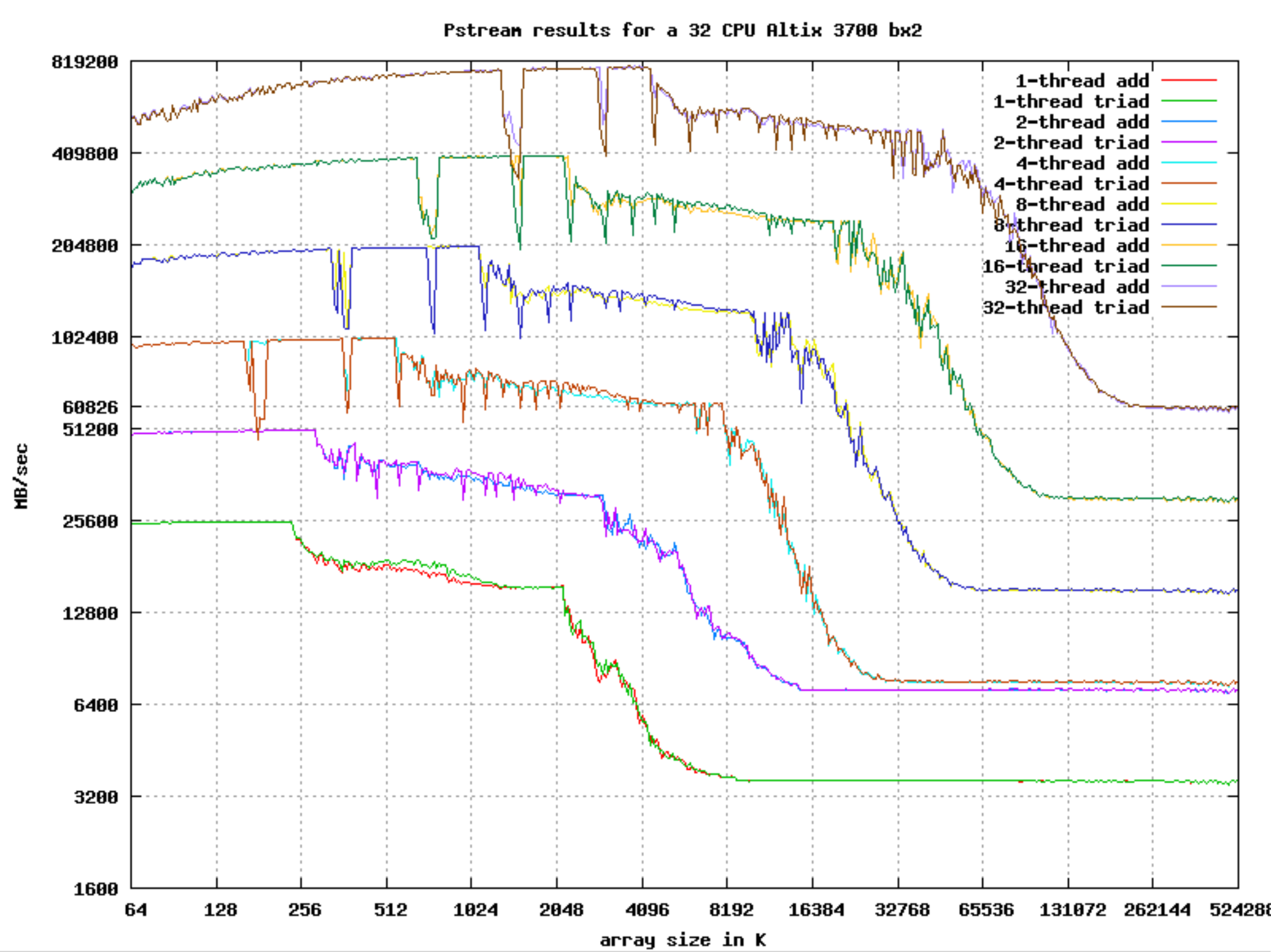
Hardware

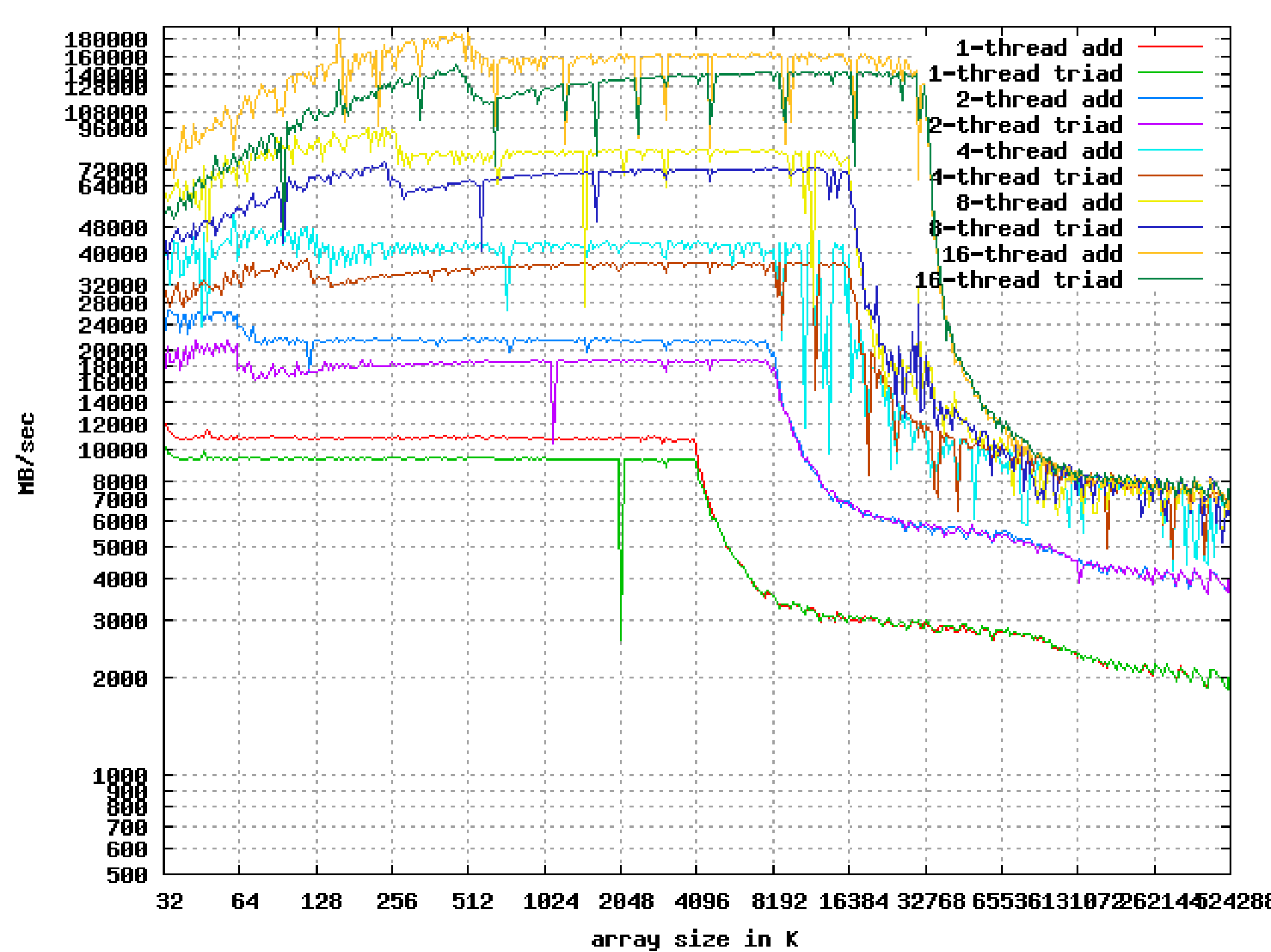
CPU Name:	AMD Opteron 8360 SE
CPU Characteristics:	
CPU MHz:	2500
FPU:	Integrated
CPU(s) enabled:	16 cores, 4 chips, 4 cores/chip
CPU(s) orderable:	2,4 chips
Primary Cache:	64 KB I + 64 KB D on chip per core
Secondary Cache:	512 KB I+D on chip per core

Software

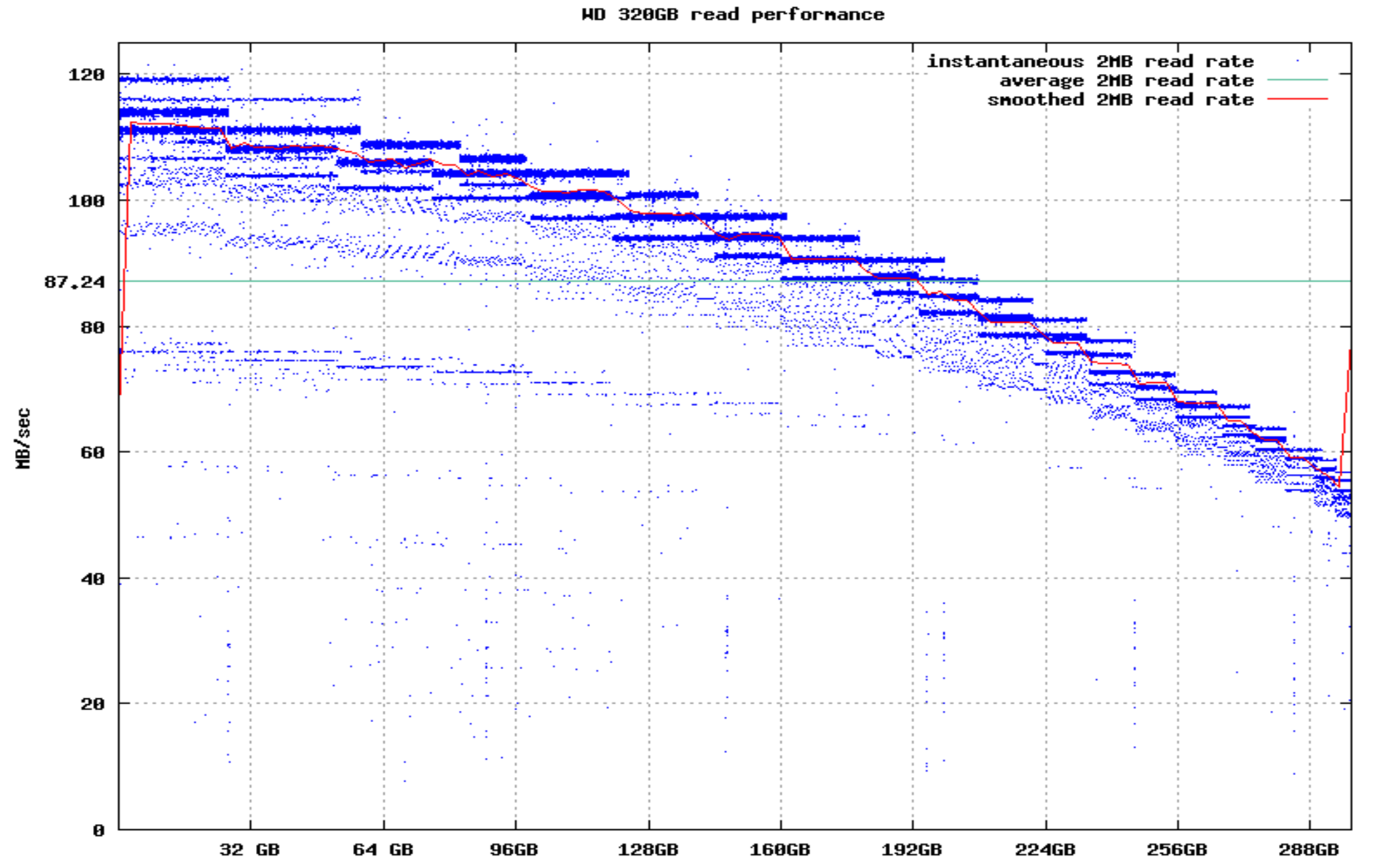
Operating System:	SuSE Linux Enterprise Server 10 (x86_64) SP1, Kernel 2.6.16.46-0.12-smp
Compiler:	PGI Server Complete Version 7.2 PathScale Compiler Suite Version 3.1
Auto Parallel:	No
File System:	ReiserFS
System State:	Run Level 3 (multi-user)
Base Pointers:	64-bit
Peak Pointers:	32/64-bit


1-thread add
1-thread triad
2-thread add
2-thread triad
4-thread add
4-thread triad
8-thread add
8-thread triad
MB/sec
array size in K
180000
160000
140000
128000
108000
96000
72000
64000
48000
40000
32000
28000
24000
20000
16000
14000
12000
10000
8000
7000
6000
5000
4000
3000
2000
1000
900
800
700
600
500
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288


Pstream results for a 32 CPU Altix 3700 bx2
MB/sec
array size in K
819200
409800
204800
102400
60826
51200
25600
12800
6400
3200
1600
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
1-thread add
1-thread triad
2-thread add
2-thread triad
4-thread add
4-thread triad
8-thread add
8-thread triad
16-thread add
16-thread triad
32-thread add
32-thread triad


1-thread add
1-thread triad
2-thread add
2-thread triad
4-thread add
4-thread triad
8-thread add
8-thread triad
16-thread add
16-thread triad
MB/sec
array size in K
180000
160000
140000
128000
108000
96000
72000
64000
48000
40000
32000
28000
24000
20000
18000
16000
14000
12000
10000
8000
7000
6000
5000
4000
3000
2000
1000
900
800
700
600
500
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288

Speed of a $70 WD sata disk



Questions?

Thank you for your time.

I can be reached at bill@cse.ucdavis.edu